SQL

Curso 1

Structured Query Language(SQL)

Linguagem de consultas estruturadas.

Documentos:

> Banco+de+dados+Fokus.db

Banco de dados relacionais são consulta em banco de dados que se conversam.

SGBD (Sistema gerencial de Banco de dados)

Nesse sentido, as principais características de um Sistema de Gerenciamento de Banco de Dados são:

1. **Armazenamento Estruturado:** Os SGBDs são projetados para armazenar dados de forma estruturada, o que significa que os dados são organizados em tabelas, com colunas que representam tipos específicos de informações. Essa estrutura facilita a busca e a recuperação de dados.
2. **Segurança:** Os SGBDs oferecem recursos avançados de segurança, incluindo controle de acesso, autenticação e criptografia, para proteger os dados contra acesso não autorizado.
3. **Recuperação de Dados:** Eles fornecem mecanismos para recuperar dados de maneira rápida e eficiente, usando consultas que permitem localizar informações específicas com facilidade.

4. **Concorrência:** Os SGBDs são projetados para permitir o acesso concorrente aos dados, o que significa que várias pessoas ou aplicativos podem acessar e modificar os dados ao mesmo tempo, sem os corromper.
5. **Integridade dos Dados:** Garantem a integridade dos dados por meio de restrições e validações, evitando a entrada de informações inconsistentes ou inválidas no banco de dados.
6. **Recuperação de Falhas:** Os SGBDs incluem mecanismos de recuperação que garantem que os dados não sejam perdidos em caso de falhas no sistema, como panes de hardware.
7. **Escalabilidade:** Permitem que os sistemas cresçam à medida que a quantidade de dados e o número de usuários aumentam, sem comprometer o desempenho.
8. **Suporte a Múltiplos Usuários:** Podem atender a várias solicitações de usuários simultaneamente, gerenciando transações de forma eficiente.
9. **Backup e Restauração:** Facilitam a criação de cópias de segurança regulares dos dados, permitindo a restauração em caso de perda de dados.

Alguns tipos de SGBD

1. **MySQL:** Um SGBD de código aberto amplamente utilizado, conhecido por sua velocidade e confiabilidade. É comumente usado em aplicativos da web e é compatível com várias linguagens de programação.
2. **Oracle Database:** Um SGBD comercial poderoso frequentemente usado em empresas para gerenciar grandes volumes de dados. Ele oferece recursos avançados de segurança, escalabilidade e recuperação.
3. **Microsoft SQL Server:** Desenvolvido pela Microsoft, é uma escolha popular para aplicativos Windows e é amplamente utilizado em ambientes corporativos. Oferece integração com tecnologias Microsoft.
4. **PostgreSQL:** Outro SGBD de código aberto, conhecido por sua extensibilidade e suporte a recursos avançados. É usado em uma ampla gama de aplicativos, incluindo sistemas geoespaciais.
5. **SQLite:** Um SGBD incorporado que é leve e não requer um servidor separado. É comumente usado em aplicativos móveis e navegadores da web.

Comandos:

Select:

SELECT (dado que deseja selecionar) FROM (Tabela desejada);

```
SELECT * FROM tabela_fornecedores;
```

Serve para mostrar todos os dados de determinada tabela.

Pesquisando dado específico com repetição.

SELECT (dado que deseja selecionar)

FROM (nome da tabela)

WHERE (nome da coluna) = (dado que desejamos consultar);

```
SELECT *
FROM tabelafornecedores
where país_de_origem = 'China';
```

Filtro específico da tabela porem pode ter dado repetido.

DISTINCT:

SELECT DISTINCT nome da coluna

FROM nome da tabela;

```
SELECT DISTINCT cliente
FROM tabelapedidos;
```

Seleciona os dados sem repetir as informações.

Pesquisando dado específico sem repetição

SELECT DISTINCT (dado que deseja selecionar)

FROM (nome da tabela)

WHERE (nome da coluna) = (dado que desejamos consultar);

```
SELECT DISTINCT Cliente
FROM tabelapedidos
where status = 'Entregue';
```

Vai fazer a pesquisa porem sem repetir com filtro específico.

> WHERE só serve para registros unicos.

CREATE TABLE

CREATE TABLE /*Nome da tabela*/ (nome_variavel tipo_variavel)

```
CREATE TABLE tabelaclientes(
    ID_Cliente INT PRIMARY KEY,
    Nome_Cliente VARCHAR(250),
    Informações_Contato VARCHAR(250)
);
```

Serve para criar uma tabela que é a estrutura que vai ser usada para armazenar os dados.

Tipos de dados mais comunSQL s

1 - **Texto (String):**

- **CHAR:** Armazena strings de tamanho fixo. Usado quando os valores têm um comprimento constante.
- **VARCHAR:** Armazena strings de tamanho variável. Apropriado para valores com comprimentos variáveis.
- **TEXTO (TEXT):** Armazena strings muito longas, como documentos ou descrições.

2 - **Numérico:**

- **INTEGER (INT):** Armazena números inteiros.
- **FLOAT:** Armazena números de ponto flutuante, geralmente usados para valores com casas decimais.

- **NUMERIC (DECIMAL):** Armazena números com uma precisão específica, geralmente usados em aplicações financeiras.

3 - **Data e Hora:**

- **DATE:** Armazena datas sem informações de horário.
- **TIME:** Armazena informações de horário.
- **TIMESTAMP:** Combina data e horário em um único tipo.

4 - **Booleano:**

- **BOOLEAN (BOOL):** Armazena valores verdadeiros ou falsos.

5 - **Binário:**

- **BLOB (Binary Large Object):** Armazena dados binários, como imagens, vídeos ou arquivos.
- **BIT:** Armazena valores binários, como 0 ou 1.

Diferença de Banco de dados e Schema.

Banco de Dados:

- Um banco de dados é uma coleção de dados organizados e relacionados.
- Ele atua como um contêiner para todos os objetos relacionados a dados, como tabelas, índices, procedimentos armazenados, visões e esquemas.
- Pode conter múltiplos esquemas, usados para organizar objetos de banco de dados.
- É uma entidade de nível superior que armazena e gerência informações em um sistema de gerenciamento de banco de dados (SGBD).

Esquema (Schema):

- Um esquema é um contêiner lógico para objetos de banco de dados, como tabelas, visões, procedimentos armazenados, etc.
- É usado para organizar e segmentar objetos em um banco de dados.
- Vários esquemas podem existir em um único banco de dados, permitindo a separação de objetos e a aplicação de permissões específicas em cada esquema.
- É uma estrutura de nível inferior em relação ao banco de dados.

DROP

DROP TABLE (Nome da Tabela);

```
DROP TABLE tabelaclientes;
#Apagando a tabela selecionada
```

Serve para apagar uma tabela específica.

DROP DATABASE (Nome do Banco De Dados)

```
DROP DATABASE Colégio_São_Paulo;
```

Serve para apagar um banco de dados organizado.

DROP SCHEMA (Nome do Schema)

```
DROP SCHEMA Turno_da_manhã;
```

Serve para apagar um schema.

ALTER

ALTER TABLE (Nome da Tabela)

ADD (nome_da_coluna) (tipo_de_dado);

```
ALTER TABLE Estudantes
ADD Idade INT;
```

Serve para alterar uma tabela específica adicionando uma nova coluna.

ALTER TABLE (Nome da Tabela)

DROP COLUMN (nome_da_coluna);

```
ALTER TABLE Estudantes
```

```
DROP COLUMN Idade;
```

Serve para alterar uma tabela específica deletando uma coluna.

Chave Primaria

Nome_da_Variavel Tipo_da_Variável PRIMARY KEY

```
CREATE TABLE tabela_Produtos(
    ID_Produto INT PRIMARY KEY
   );
```

São um tipo de dado unico que não pode ser repetido e não pode ser nulo.

Chave estrangeira

FOREIGN KEY (Nome_da_coluna) REFERENCES Tabela_desejada (Coluna_referenciada)

```
CREATE TABLE tabela_Produtos(
    ID_Produto INT PRIMARY KEY,
    nome_Produto VARCHAR(250),
    Descricao_Produto TEXT,
    categoria_Produto INT,
    proco_Compra_Produto DECIMAL(10,2),
    unidade_Produto VARCHAR(50),
    fornecedor_Produto INT,
    data_Inclusao_Produto DATE,
    FOREIGN KEY (categoria) REFERENCES Tabela_Categorias (id_Categori
a),
    FOREIGN KEY (fornecedor) REFERENCES Tabela_Fornecedores (ID)
);
```

São responsáveis por relacionar as tabelas existentes.

Inserindo Um Valor Na Tabela

INSERT INTO Nome_Tabela(

Dados_desejados

)

Values

((Dados1), (Dados2), (Dados3));

```
INSERT INTO tabelaclientes(
id_cliente,
nome_cliente,
informações_contato,
endereço_cliente
)
VALUES
('1','Ana Silva', 'ana.silva@email.com', 'Rua Flores - Casa 1');
```

Sempre que precisar adicionar uma chave primária mesmo que seja um “int” é necessário colocar entre ‘’(Aspas Simples)

Inserindo Alguns Valores Na Tabela

INSERT INTO Nome_Tabela(

Dados_desejados

)

Values

(Dados1), (Dados2), (Dados3),

(Dados4), (Dados5), (Dados6),

(Dados7), (Dados8), (Dados9);

```
INSERT INTO tabelaprodutos
(ID_Produto,
  nome_Produto,
  Descricao_Produto,
  categoria_Produto,
  proco_Compra_Produto,
  unidade_Produto,
  fornecedor_Produto,
```

```
  data_Inclusao_Produto)
  VALUES
  (1, 'Smartphone X', 'Smartphone de última geração', 1, 699.99, 'Unidade',
1, '2023-08-01'),
  (2, 'Notebook Pro', 'Notebook poderoso com tela HD', 2, 699.99, 'Unidad
e', 2, '2023-08-02'),
  (3, 'Tablet Lite', 'Tablet compacto e leve', 3, 299.99, 'Unidade', 3, '2023-0
8-03'),
  (4, 'TV LED 55', 'TV LED Full HD de 55 polegadas', 4, 599.99, 'Unidade', 4,
'2023-08-04'),
  (5, 'Câmera DSLR', 'Câmera digital DSLR com lente intercambiável', 5, 69
9.99, 'Unidade', 5, '2023-08-05'),
  (6, 'Impressora Laser', 'Impressora laser de alta qualidade', 6, 349.99, 'Uni
dade', 6, '2023-08-06'),
  (7, 'Mouse Óptico', 'Mouse óptico sem fio', 7, 19.99, 'Unidade', 7, '2023-08
-07'),
  (8, 'Teclado sem Fio', 'Teclado sem fio ergonômico', 8, 39.99, 'Unidade',
8, '2023-08-08');
```

Inserindo várias informações na tabela desejada e as informações têm que seguir o padrão escrito na parte do insert.

PRAGMA

PRAGMA table_info(/*tabela desejada*/);

```
PRAGMA table_info(tabelapedidos);
```

mostrará os tipos de dados das colunas.

Inserindo dados pelo Select

INSERT INTO /*Tabela de recebimento desejada*/

(

/*dados desejados a ser recebido*/)

SELECT

/*dados desejados a ser enviado*/

```
PRAGMA table_info(tabelapedidos);
INSERT INTO tabelapedidosgold
(ID_Pedido_Gold,
Data_Do_Pedido_Gold,
Status_Gold,
Total_Do_Pedido_Gold,
Cliente_Gold,
Data_De_Envio_Estimada_Gold)
--No select vai ser necessario alterar o tipo do c4 de texto para numero re
al, para pois  isso vai ajudar a fazer a verificação.
SELECT
c1, -- ID_Pedido_Gold
c2, -- Data_Do_Pedido_Gold
c3, -- Status_Gold
CAST(c4 AS REAL), -- Total_Do_Pedido_Gold convertido para número
c5, -- Cliente_Gold
c6  -- Data_De_Envio_Estimada_Gold
FROM tabelapedidos
WHERE CAST(c4 AS REAL) >= 400;
--renomeando as colunas das tabelas através de nomes alternativos.
SELECT
c1 AS ID_Pedido_Gold,
c2 AS Data_Do_Pedido_Gold,
c3 AS Status_Gold,
c4 AS Total_Do_Pedido_Gold,
c5 AS Cliente_Gold,
c6 AS Data_De_Envio_Estimada_Gold
FROM tabelapedidos
WHERE CAST(c4 AS REAL) >= 400;
--Alterando na tabela o c4 para formato numero
ALTER TABLE tabelapedidos
ADD COLUMN c4_numeric REAL;
```

```
UPDATE tabelapedidos
SET c4_numeric = CAST(c4 AS REAL);
--Atualizando na tabelapedidos que o c4 vai ser um numero real agora.
```

```
SELECT * FROM tabelapedidosgold;
--Verificando todos os dados que foram selecionados estão corretos
```

Exemplos de insert com select

A combinação do INSERT com SELECT permite inserir dados em uma tabela baseados em uma seleção de dados de outra ou da mesma tabela. Esta combinação é bastante útil quando desejamos duplicar dados, migrar informação entre tabelas ou criar backups. Vamos realizar um aprofundamento no uso combinado desses comandos para melhor entendimento.

A sintaxe básica da combinação de INSERT e SELECT é:

```
INSERTINTO tabela_destino (coluna1, coluna2, ... )
SELECT coluna1, coluna2, ...
FROM tabela_origem
WHERE condição;
```

Aqui, `tabela_destino` é a tabela onde você quer inserir os dados e `tabela_origem` é a tabela onde os dados estão originalmente. As `coluna1, coluna2,...` precisam ser as mesmas tanto na tabela de destino quanto na de origem. A cláusula WHERE é opcional e permite especificar condições para a seleção dos dados.

1 - **Duplicação de dados de uma tabela**

Suponha que temos uma tabela chamada Alunos, com as colunas ID, nome e idade, e queremos copiar todos os dados para uma nova tabela chamada Backup_Alunos. Para isso, usamos o seguinte comando:

```
INSERTINTO Backup_Alunos (ID, nome, idade)
SELECT ID, nome, idade
FROM Alunos;
```

2 - **Transferência de dados entre tabelas**

Supondo agora que tenhamos duas tabelas, Alunos e Ex_Alunos, ambas com as colunas ID, nome e idade. Queremos mover os alunos com mais de 20 anos

para a tabela Ex_Alunos. Para isso, primeiramente, vamos inserir esses alunos na tabela Ex_Alunos:

```
INSERTINTO Ex_Alunos (ID, nome, idade)
SELECT ID, nome, idade
FROM Alunos
WHERE idade > 20;
```

Em seguida, precisaríamos remover esses alunos da tabela Alunos, o que é feito com o comando DELETE:

```
DELETE FROM Alunos
WHERE idade > 20;
```

3 - **Inserção condicional de dados**

Imagine que temos uma tabela de Vendas, com as colunas ID_venda, ID_produto e quantidade, e uma tabela de Estoque, com as colunas ID_produto e quantidade. Queremos diminuir a quantidade em estoque dos produtos vendidos. Para isso, poderíamos usar a seguinte combinação de INSERT, SELECT e UPDATE:

```
INSERTOR REPLACEINTO Estoque (ID_produto, quantidade)
SELECT Vendas.ID_produto, Estoque.quantidade - Vendas.quantidade
FROM Vendas
JOIN EstoqueON Vendas.ID_produto = Estoque.ID_produto;
```

Operadores Lógicos Matemáticos

> > Comparação de maior que

> < Comparação de menor que

> = Igual a

> != (diferente de)

> >= e <= maior igual e menor igual

> AND Fazer comparações simultâneas de múltiplos valores

> OR Fazer uma comparação de ou uma condição, ou outra

> NOT Negação de alguma condição

> Like para padrões de texto

Filtros simples

SELECT * FROM /*Tabela desejada*/ Where /*Variavel desejada*/ /*Operador logico*/ /*Condição*/

```
SELECT * FROM tabelapedidos where c2 > '2023-09-19';
```

Filtros Compostos

AND

SELECT * FROM /*Tabela desejada*/ Where /*Variavel desejada*/ /*Operador logico*/ /*Condição*/ AND /*Variavel desejada*/ /*Operador logico*/ /*Condição*/;

```
SELECT * FROM tabelapedidos where c4 >= 200.00 AND c3 = 'Pendente';
```

Serve para quando queremos buscar com múltiplas variedades simultaneamente.

OR

SELECT * FROM /*Tabela desejada*/ Where /*Variavel desejada*/ /*Operador logico*/ /*Condição*/ OR /*Variavel desejada*/ /*Operador logico*/ /*Condição*/;

```
SELECT * FROM tabelapedidos where c3 = 'Processando' OR c3 = 'Penden
te';
```

Serve para quando queremos buscar que atenda somente uma das variáveis selecionadas.

NOT

SELECT * FROM /*Tabela desejada*/ Where NOT /*Variavel desejada*/ /*Operador logico*/ /*Condição*/

```
SELECT * FROM tabelapedidos where NOT c3 = 'Pendente';
```

Deseja ver a tabela sem determinada variavel.

BETWEEN

SELECT * FROM /*Tabela desejada*/ Where /*Variavel desejada*/ BETWEEN /*Condição*/ AND /*Condição*/;

```
SELECT * FROM tabelapedidos WHERE c6 BETWEEN '2023-08-01' AND '2
023-09-01';
```

Quando se deseja ver determinada dado entre determinada condição.

Order BY(ASC)

SELECT * FROM /*Tabela desejada*/ Where /*Variavel desejada*/ ORDER BY /*Coluna que vai servir para ser ordenada*/

```
SELECT * FROM tabelaprodutos WHERE preco_compra_produto BETWEEN
200 AND 600 ORDER BY nome_produto;
```

Serve para ordernar a tabela através de determinada coluna desejada.

E por padrão vem em forma ascendente.

ORDER BY ... DESC

SELECT * FROM /*Tabela desejada*/ Where /*Variavel desejada*/ ORDER BY /*Coluna que vai servir para ser ordenada* DESC/

```
SELECT * FROM tabelaprodutos WHERE preco_compra_produto BETWEEN 200 AND 600 ORDER BY fornecedor_produto DESC;
```

Deve para ordernar a coluna em ordem contraria do padrão (Coluna crescente vira decrescente).

ALIAS

SELECT /*Coluna Desejada*/ AS /* Apelido desejado */ FROM /*Tabela desejada*/;

```
SELECT informacoes_contato AS email_Cliente FROM tabelaclientes;
```

Serve para colocar apelidos para alguma coluna assim sem ter a necessidade de alterar o nome da tabela.

UPDATE

UPDATE /*Tabela que deseja alterar*/
SET /*Coluna Desejada*/ =/*Dado que vai ser inserido*/
WHERE /*Coluna Desejada*/ = /*Dado que vai ser alterado*/;

```
UPDATE tabelapedidos
SET c3 = 'Enviado'
WHERE c3 = 'Processando';
```

Serve para alterarmos determinado dado da tabela.

DELETE

DELETE FROM /*Tabela desejada*/

WHERE /*Variavel desejada*/ /*Operador logico*/ /*Condição*/;

```
DELETE FROM tabelafornecedores
where c3 = 'Turquia';
```

```
DELETE FROM tabelafornecedores
WHERE c1 > 35;
```

Serve para deletar parte especifica do banco de dados.

DELETE CASCADE

```
CREATETABLE Clientes (
    IDINTPRIMARY KEY,
    NomeVARCHAR(50)
);

CREATETABLE Pedidos (
    PedidoIDINTPRIMARY KEY,
    ClienteIDINT,
    DescricaoVARCHAR(100),
FOREIGN KEY (ClienteID)REFERENCES Clientes(ID)ON DELETE CASCADE
);
```

ON DELETE CASCADE, você garante que, se um registro na tabela "pai" for excluído, todos os registros relacionados nas tabelas "filhas" serão automaticamente excluídos, evitando referências a registros ausentes.

Curso 2

ORDER BY

select * FROM tabela_desejada
ORDER BY Coluna_desejada;

```
select * FROM HistoricoEmprego
```

```
ORDER BY salario DESC;
```

Serve para ordernar alguma tabela baseando-se em uma ordem crescente.

> 💡 Se adicionarmos o DESC na frente a leitura vai ser de forma decrescente

LIMIT

select * FROM tabela_desejada
ORDER BY Coluna_desejada
LIMIT Quantidade_desejada;

```
select * FROM HistoricoEmprego
ORDER BY salario DESC
LIMIT 5;
```

Serve para limitar uma quantidade de dados desejada, como no exemplo a cima estou limitando a quantidade de resgistros que deseja ser visualizado seja 5.

TOP

SELECT TOP número_colunas FROM tabela;

```
SELECT * FROM clientes
LIMIT 10 OFFSET 10;
```

Serve para limitar uma quantidade de dados desejada, como no exemplo a cima estou limitando a quantidade de resgistros que deseja ser visualizado seja 10.

LIMIT

select * FROM tabela_desejada

LIMIT 10 OFFSET 10

```
SELECT *FROM clientes
```

```
LIMIT 10OFFSET 10;
```

`LIMIT` é mais comum em ambientes como MySQL e PostgreSQL, `TOP` é específico para o ambiente Microsoft. Além disso, a cláusula `LIMIT` pode ser acompanhada por

> `OFFSET` para pular um número específico de linhas, uma funcionalidade que é especialmente útil para implementar paginação em aplicações web.

ISNULL

select * FROM tabela_desejada
ORDER BY Coluna_desejada isnull
LIMIT Quantidade_desejada;

```
select * FROM HistoricoEmprego
WHERE datatermino isnull
ORDER BY salario DESC
LIMIT 5;
```

Serve para identificar os dados de determinado campo que sejá null.

LIKE

select * FROM tabela_desejada

where /*Coluna Desejada*/ LIKE ‘Termo_desejado_a_ser_pesquisado_%’

```
SELECT * FROM Treinamento
where curso LIKE 'O poder%'
/*where curso LIKE '%realizar%'*/
```

Serve para pesquisar todos os dados de detrminada coluna que seja parecido com o termo desejado a ser presquisado.

> A % serve para indicar que vai ter uma serie de strings a frente ou atras do termo desejado.

```
SELECT * FROM Colaboradores
WHERE Nome LIKE'Isadora%'
```

AND, OR, IN e NOT

AND

select * FROM tabela_desejada

where /*Coluna Desejada*/ = ‘Termo_desejado_a_ser_pesquisado’ AND ‘Termo_desejado_a_ser_pesquisado’ NOTNULL

```
SELECT * FROM HistoricoEmprego
WHERE cargo = 'Professor' AND
datatermino NOTNULL;
```

Só retorna se atender todas as condições pedidas.

> No caso de cima só ira retornar se cargo seja professor e data de termino não esteja nula.

OR

select * FROM tabela_desejada

where /*Coluna Desejada*/ = ‘Termo_desejado_a_ser_pesquisado’ OR ‘Termo_desejado_a_ser_pesquisado’ NOTNULL

```
SELECT * FROM HistoricoEmprego
WHERE cargo = 'Oftalmologista'
or cargo = 'Dermatologista';
```

Serve para trazer os dados se atenderem uma das condições ou ambas sem ser exclusiva.

IN

select * FROM tabela_desejada

where /*Coluna Desejada*/ IN (‘Termo_desejado_a_ser_pesquisado’, ‘Termo_desejado_a_ser_pesquisado’ , ETC);

```
SELECT * FROM HistoricoEmprego
WHERE cargo IN ('Oftalmologista', 'Dermatologista', 'Professor');
```

Traz todos os campos de uma vez sem repetir os operadores.

NOT /*Operador desejado*/

select * FROM tabela_desejada

where /*Coluna Desejada*/ NOT /*Operador desejado*/ (‘Termo_desejado_a_ser_pesquisado’, ‘Termo_desejado_a_ser_pesquisado’ , ETC);

```
SELECT * FROM HistoricoEmprego
WHERE cargo NOT IN ('Oftalmologista', 'Dermatologista', 'Professor');
```

Traz todos os campos menos os que foram especificados.

Considerações Importantes

- A ordem das operações é fundamental: `AND` é avaliado antes de `OR`, a menos que parênteses sejam usados para modificar a ordem.
- A clareza da lógica é essencial para evitar erros, especialmente em consultas mais complexas.
- Testar as consultas com diferentes conjuntos de dados pode ajudar a garantir que a lógica aplicada está correta.

Utilizando diversos operadores

```
SELECT * FROM Treinamento
WHERE (curso LIKE 'O direito%' AND instituicao = 'da Rocha')
```

```
OR (curso LIKE 'O conforto%' AND instituicao = 'das neves');
```

O codigo indica que estamos selecionando todos os dados da tabela treinamento,

de onde o curso seja PARECIDO com o direito alguma coisa E intituição é da rocha OU

curso seja PARECIDO com o conforto alguma coisa E intituição é das neves.

Função de agregação

MAX

SELECT /*informação_coluna_desejada*/, MAX (/*coluna que vai ser aplicada a determinada função de agregação*/)

FROM /*Tabela desejada.*/

```
SELECT mes, MAX(faturamento_bruto)
from faturamento;
```

Traz o maior valor da coluna desejada.

MIN

SELECT /*informação_coluna_desejada*/, MIN (/*coluna que vai ser aplicada a determinada função de agregação*/)

FROM /*Tabela desejada.*/

```
SELECT mes, MIN(faturamento_bruto)
from faturamento;
```

Traz o menor valor da coluna desejada.

SUM

SELECT /*informação_coluna_desejada*/, SUM (/*coluna que vai ser aplicada a determinada função de agregação*/)

FROM /*Tabela desejada.*/;

```
SELECT SUM(numero_novos_clientes) AS 'Novos clientes 2023' FROM faturar
WHERE mes LIKE '%2023';
```

Faz a soma dos valores desejados.

AVG(Average = media)

SELECT AVG(/*informação_coluna_desejada*/)

FROM /*Tabela desejada.*/;

```
SELECT AVG(lucro_liquido)
FROM faturamento;
```

Serve para tirar a media dos valores que fortam requisitados.

COUNT

SELECT COUNT(/*informação_coluna_desejada*/)

FROM /*Tabela desejada.*/

WHERE/*Condição desejada*/

```
SELECT COUNT(*)
FROM HistoricoEmprego
WHERE datatermino notNULL;
```

Serve para fazer uma contagem de todos os dados de determinada tabela.

```
SELECT COUNT (*)FROM Licencas
where tipolicenca = 'férias';
```

> Tambem serve para agrupar por certo tipo de informação. coluna, etc

GROUP BY

SELECT /*informação_coluna_desejada*/, COUNT()/*Todos*/ FROM /*Tabela desejada.*/
GROUP BY /*dados_de_determinada_caracteristica*/

```
SELECT parentesco, COUNT(*) FROM Dependentes
GROUP BY parentesco;
/*Selecionando a tabela parentesco, fazendo a contagem de quantos tem de
```

Serve para agrupar os mesmos valores da coluna desejada.

> /*Selecionando a tabela parentesco, fazendo a contagem de quantos tem de determinados dependentes e agrupando eles pela caracteristicas parentescos *

```
SELECT instituicao, COUNT(curso)
FROM Treinamento
GROUP BY instituicao;
```

HAVING

SELECT /*informação_coluna_desejada*/, COUNT()/*Todos*/ FROM /*Tabela desejada.*/
GROUP BY /*dados_de_determinada_caracteristica*/

HAVING /*dado que desejamos consultar*/

```
SELECT instituicao, COUNT(curso)
FROM Treinamento
GRoup by instituicao
HAVING COUNT (curso) > 2;
```

É utilizada apos utilizar o GROUP BY.

Diferença entre WHERE e HAVING

- `WHERE` é usado para filtrar registros antes de qualquer agrupamento.

- `HAVING` é usado para filtrar grupos criados pela cláusula `GROUP BY`.

LENGTH

SELECT /*Coluna desejada*/, LENGTH(/*Coluna que a função vai ser aplicada*/) /*Nome do resultado da coluna selcionada*/

FROM /*Tabela desejada.*/

WHERE *Nome do resultado da coluna selcionada*/ = /*Dado desejado*/

```
SELECT nome, LENGTH(cpf) qtd
FROM Colaboradores
WHERE qtd = 11;
```

> Aqui trazemos todas as linhas que tem cpf com 11 digitos

```
SELECT COUNT(*), LENGTH(cpf) qtd
FROM Colaboradores
WHERE qtd = 11;
```

> Verifica quantas tatelas tem a coluna cpf com 11 digitos e faz a contagem

CONCAT

SELECT ('/*Texto desejado*/' || /*informação_coluna_ que_deseja_mostrar*/ /*Texto desejado*/'|| /*informação_coluna_ que_deseja_mostrar*/) AS texto

FROM /*Tabela desejada.*/;

```
SELECT ('A pessoa colaboradora' || nome || 'de CPF' ||cpf|| 'possui o seguinte
FROM Colaboradores;
```

> Serve para trazermos as informacões cecritas de uma forma que facilite o entendimento.

```
SELECT UPPER('A pessoa colaboradora' || nome || 'de CPF' ||cpf|| 'possui o s
FROM Colaboradores;
```

> UPPER Serve para deixar todas as letras em maiuscula

```
SELECT LOWER('A pessoa colaboradora' || nome || 'de CPF' ||cpf|| 'possui o s
FROM Colaboradores;
```

> LOWER Serve para deixar todas as letras em minusculas

Função TRIM

- **Funcionalidade:** A função `TRIM` remove espaços (ou outro conjunto especificado de caracteres) do início e do fim de uma string.
- **Sintaxe Básica:** `TRIM(string, [caractere_para_trimar])`
- **Exemplo de Uso:** Para remover espaços do início e do fim da coluna `nome`:

```
SELECT TRIM(nome)FROM tabela;
Copiar código
```

Função INSTR

- **Funcionalidade:** `INSTR` retorna a posição de uma substring dentro de uma string. Equivalente ao `CHARINDEX` em alguns outros sistemas.
- **Sintaxe Básica:** `INSTR(string, substring)`
- **Exemplo de Uso:** Para encontrar a posição da substring 'abc' dentro da coluna `descricao`:

```
SELECT INSTR(descricao, 'abc')FROM tabela;
Copiar código
```

Isso retornará um número indicando a posição inicial de 'abc' em `descricao`, ou 0 se 'abc' não for encontrado.

Função REPLACE

- **Funcionalidade:** `REPLACE` substitui todas as ocorrências de uma substring específica por outra substring dentro de uma string.
- **Sintaxe Básica:** `REPLACE(string, substring_a_substituir, substring_para_substituir)`
- **Exemplo de Uso:** Para substituir 'hello' por 'hi' na coluna `saudacao`:

```
SELECT REPLACE(saudacao, 'hello', 'hi')FROM tabela;
Copiar código
```

Função SUBSTR (ou SUBSTRING em alguns sistemas)

- **Funcionalidade:** `SUBSTR` extrai uma parte de uma string com base em um ponto de início e um comprimento especificados.
- **Sintaxe Básica:** `SUBSTR(string, inicio[, comprimento])`
- **Exemplo de Uso:** Para extrair os primeiros 5 caracteres da coluna `comentario`:

```
SELECT SUBSTR(comentario, 1, 5)FROM tabela;
Copiar código
```

Se `comprimento` não for especificado, `SUBSTR` retornará todos os caracteres a partir da posição `inicio` até o final da string.

Considerações Importantes

- Ao trabalhar com `TRIM`, se nenhum caractere específico for fornecido para remoção, ele removerá espaços por padrão.
- A função `INSTR` é particularmente útil para localizar substrings e pode ser usada em operações mais complexas, como extrações condicionais ou verificação de presença de padrões.
- `REPLACE` é uma ferramenta poderosa para limpeza e formatação de dados, sendo capaz de alterar padrões específicos em uma grande quantidade de texto.

- `SUBSTR` é amplamente utilizada para cortar e analisar partes de strings, especialmente quando combinada com outras funções como `INSTR`.

Data

SELECT /*Coluna desejada*/, STRFTIME('%/*Modelo de dada desejada*/', /*Coluna desejada*/)

FROM /*Tabela desejada*/;

```
SELECT id_colaborador, STRFTIME('%Y/%M', datainicio)
FROM Licencas;
```

> Serve para organizarmos a forma que a data vai ser exibida.

JULIANDAY

SELECT /*Coluna desejada*/, JULIANDAY(/*Coluna desejada*/) - JULIANDAY(/*Coluna desejada*/)

FROM /*Tabela desejada*/

WHERE /*Coluna Especifica*/ IS NOT NULL;

```
SELECT id_colaborador, JULIANDAY(datatermino) - JULIANDAY (datacontrata
FROM HistoricoEmprego
where datatermino is NOT NULL;
```

> Serve para calcularmos determinada diferença entre as datas no qual retira todos os dados a onde data termino seria nula.

Função DATE

- **Funcionalidade:** A função `DATE` é usada para extrair a data de um valor de data e hora ou para obter a data atual. Ela retorna a data no formato 'YYYY-MM-DD'.

- **Sintaxe Básica:** `DATE('now', '[modificador]')`
- **Exemplo de Uso:** Para obter a data atual:

```
SELECTDATE('now');
Copiar código
```

Para obter a data 10 dias atrás:

```
SELECTDATE('now', '-10 days');
Copiar código
```

Função TIME

- **Funcionalidade:** A função `TIME` é usada para extrair a hora de um valor de data e hora ou para obter a hora atual. Ela retorna a hora no formato 'HH:MM:SS'.
- **Sintaxe Básica:** `TIME('now', '[modificador]')`
- **Exemplo de Uso:** Para obter a hora atual:

```
SELECTTIME('now');
Copiar código
```

Função DATETIME

- **Funcionalidade:** `DATETIME` é uma função mais abrangente que retorna tanto a data quanto a hora no formato 'YYYY-MM-DD HH:MM:SS'. Pode ser usada para obter o momento atual ou converter/modificar valores de data e hora existentes.
- **Sintaxe Básica:** `DATETIME('now', '[modificador]')`
- **Exemplo de Uso:** Para obter a data e hora atuais:

```
SELECT DATETIME('now');
Copiar código
```

Para obter a data e hora exatas 1 ano no futuro:

```
SELECT DATETIME('now', '+1 year');
Copiar código
```

Função CURRENT_TIMESTAMP

- **Funcionalidade:** `CURRENT_TIMESTAMP` é uma função de conveniência que retorna a data e hora atuais no formato 'YYYY-MM-DD HH:MM:SS'. É equivalente a usar `DATETIME('now')`.
- **Sintaxe Básica:** `CURRENT_TIMESTAMP`
- **Exemplo de Uso:** Para obter o timestamp atual:

```
SELECT CURRENT_TIMESTAMP;
Copiar código
```

Considerações Importantes

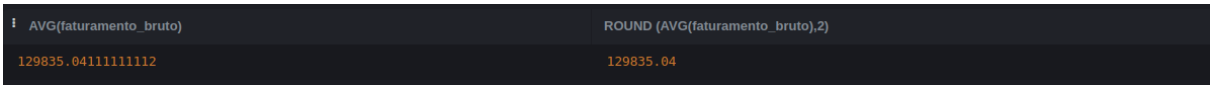
- Os modificadores, como '-10 days' ou '+1 year', são usados para ajustar a data/hora retornada. Eles podem ser combinados para representar períodos específicos de tempo.
- Essas funções são extremamente úteis para gerar e manipular dados de data e hora, permitindo cálculos temporais, conversões e a extração de componentes específicos.
- O conhecimento preciso de como as datas e horas são armazenadas e manipuladas em seu sistema de banco de dados é crucial para utilizar essas funções efetivamente e evitar erros comuns relacionados a fusos horários e formatos.

ROUND

```
SELECT AVG(/*coluna_desejada*/),ROUND
(AVG(/*coluna_desejada*/),/*numero_casas_desejadas*/)
FROM /*tabela_desejada*/;
```

```
SELECT AVG(faturamento_bruto),ROUND (AVG(faturamento_bruto),2)
FROM faturamento;
```

> Serve para arredonda um número de casas decimais determinados por nós.



AVG(faturamento_bruto)	ROUND (AVG(faturamento_bruto),2)
129835.04111111112	129835.04

CEIL

SELECT CEIL(/*coluna_desejada*/),CEIL (/*coluna_desejada*/)
FROM /*tabela_desejada*/;

```
SELECT CEIL(faturamento_bruto),CEIL (despesas)
FROM faturamento;
```

> `CEIL`, que arredonda valores numéricos **para cima**, ou seja, para o próximo número inteiro de valor maior;

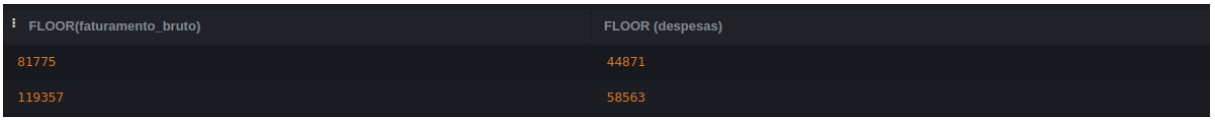
CEIL(faturamento_bruto)	CEIL (despesas)
81776	44872
119358	58564

FLOOR

SELECT FLOOR(/*coluna_desejada*/),FLOOR (/*coluna_desejada*/)
FROM /*tabela_desejada*/;

```
SELECT FLOOR(faturamento_bruto),FLOOR (despesas)
FROM faturamento;
```

> `FLOOR`, que arredonda para baixo, para o inteiro menor mais próximo;



FLOOR(faturamento_bruto)	FLOOR (despesas)
81775	44871
119357	58563

Função POWER

- **Funcionalidade:** `POWER` é usada para elevar um número a uma potência específica.
- **Sintaxe Básica:** `POWER(base, expoente)`
- **Exemplo de Uso:** Para elevar 2 à 3ª potência:

```
SELECT POWER(2, 3);
Copiar código
```

Isso retornará 8, que é 2^3.

Função SQRT

- **Funcionalidade:** `SQRT` retorna a raiz quadrada de um número.
- **Sintaxe Básica:** `SQRT(numero)`
- **Exemplo de Uso:** Para encontrar a raiz quadrada de 16:

```
SELECT SQRT(16);
Copiar código
```

Isso retornará 4, que é a raiz quadrada de 16.

Função RANDOM

- **Funcionalidade:** `RANDOM` gera um número inteiro aleatório entre -9223372036854775808 e +9223372036854775807.
- **Sintaxe Básica:** `RANDOM()`
- **Exemplo de Uso:** Para gerar um número aleatório:

```
SELECT RANDOM();
Copiar código
```

Cada chamada retornará um número inteiro aleatório diferente.

Função ABS

- **Funcionalidade:** `ABS` retorna o valor absoluto de um número, que é o número sem seu sinal.
- **Sintaxe Básica:** `ABS(numero)`

- **Exemplo de Uso:** Para obter o valor absoluto de -5:

```
SELECT ABS(-5);
Copiar código
```

Isso retornará 5.

Função HEX

- **Funcionalidade:** `HEX` converte um número ou uma string para a sua forma hexadecimal.
- **Sintaxe Básica:** `HEX(string)`
- **Exemplo de Uso:** Para converter 255 para hexadecimal:

Isso retornará 'FF'. E para converter a string 'hello':

```
SELECT HEX('hello');
Copiar código
```

Isso retornará '68656C6C6F', que é a representação hexadecimal da string 'hello'.

Considerações Importantes

- `POWER` e `SQRT` são particularmente úteis para cálculos científicos e financeiros.
- `RANDOM` é útil para situações onde você precisa de dados aleatórios, como na criação de amostras ou em simulações.
- `ABS` é frequentemente usado em análises matemáticas e estatísticas para garantir que apenas a magnitude de um número seja considerada.
- `HEX` é útil para trabalhos com sistemas que usam representações hexadecimais, como trabalhos com cores na web ou com dados binários.

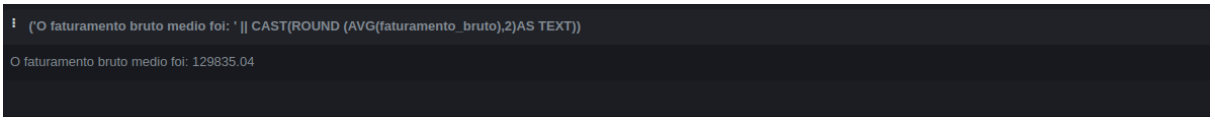
CAST

SELECT ('/*Texto desejado*/' || CAST(ROUND (AVG(/*Coluna_Desejada*/, *numero_casas_desejadas*/)/*Como deseja que apareça*/))

FROM /*tabela_desejada*/;

```
SELECT ('O faturamento bruto medio foi: ' || CAST(ROUND (AVG(faturamento_
FROM faturamento;
```

> `CAST`, que transforma um tipo de dado em outro tipo de dado para usá-lo em uma operação, comando ou função específica e conseguir extrair a informação desejada.



1. CAST

- **Funcionalidade:** Converte um tipo de dados de uma expressão para outro tipo especificado.
- **SGBDs Compatíveis:** Quase todos os SGBDs principais, incluindo MySQL, PostgreSQL, SQL Server, SQLite e Oracle. Única função de conversão disponível no **SQLite online**.
- **Sintaxe:** `CAST(expressao AS tipo)`

2. CONVERT

- **Funcionalidade:** Semelhante ao `CAST`, mas com uma sintaxe ligeiramente diferente e, em alguns SGBDs, recursos adicionais.
- **SGBDs Compatíveis:** Principalmente SQL Server e MySQL. A função `CONVERT` no MySQL é usada mais comumente para conversão de codificação de caracteres, não tipos de dados.
- **Sintaxe (SQL Server):** `CONVERT(tipo, expressao [, estilo])`

3. TO_NUMBER, TO_CHAR, TO_DATE (Funções específicas do Oracle)

- **Funcionalidade:** Converte strings para números (`TO_NUMBER`), números ou datas para strings (`TO_CHAR`), e strings para datas (`TO_DATE`).
- **SGBDs Compatíveis:** Oracle.

- **Sintaxe:** `TO_NUMBER(string [, formato [, 'nlsparam']])` , `TO_CHAR(valor [, formato [, 'nlsparam']])` , `TO_DATE(string [, formato [, 'nlsparam']])`

4. PARSE, TRY_PARSE, TRY_CONVERT (SQL Server)

- **Funcionalidade:** `PARSE` tenta converter uma string para um tipo de dados numérico ou de data/hora com um estilo de cultura opcional. `TRY_PARSE` e `TRY_CONVERT` são versões mais seguras que retornam NULL em vez de um erro se a conversão falhar.
- **SGBDs Compatíveis:** SQL Server.
- **Sintaxe:** `PARSE(string AS tipo USING cultura)` , `TRY_PARSE(string AS tipo USING cultura)` , `TRY_CONVERT(tipo, expressao [, estilo])`

5. STR_TO_DATE (MySQL)

- **Funcionalidade:** Converte uma string em um formato de data especificado para uma data.
- **SGBDs Compatíveis:** MySQL.
- **Sintaxe:** `STR_TO_DATE(string, formato)`

6. TO_NUMBER, TO_CHAR (PostgreSQL)

- **Funcionalidade:** `TO_NUMBER` converte uma string para um número, e `TO_CHAR` converte um número ou data para uma string, ambos com base em um formato especificado.
- **SGBDs Compatíveis:** PostgreSQL.
- **Sintaxe:** `TO_NUMBER(string, formato)` , `TO_CHAR(valor, formato)`

Considerações Importantes:

- **Compatibilidade:** Sempre verifique a documentação específica do seu SGBD para entender a disponibilidade e o uso exato de cada função, pois pode haver pequenas variações na sintaxe e no comportamento.
- **Uso Cuidadoso:** A conversão de tipos de dados deve ser feita com cuidado, especialmente ao converter entre numéricos e strings ou ao lidar com datas, para evitar erros ou resultados inesperados.
- **Dependência da Versão:** Alguns SGBDs podem adicionar, modificar ou depreciar funções em diferentes versões, então é importante considerar a

versão específica que você está usando.

Case

SELECT /*Coluna_desejada*/,/*Coluna_desejada*/,

CASE

WHEN /*Condicional*/ THEN /*ação_Desejada*/

WHEN /*Condicional*/ BETWEEN /*1 Valor*/ AND /*2 Valor*/ THEN /*ação_Desejada*/

ELSE /*Condição caso não entre em nenhum dos acima*/

END AS /*Nome da tabela desejada*/

FROM /*Tabela_desejada*/

```
SELECT id_colaborador, cargo, salario,
CASE
WHEN salario < 3000 THEN 'Baixo'
WHEN salario BETWEEN 3000 AND 6000 THEN 'Medio'
ELSE 'Alto'
END AS categoria_salario
FROM HistoricoEmprego;
```

> `case()` que usamos para criar condições para a nossa consulta, determinando que, caso uma condição específica seja atendida, o resultado deve vir de uma forma, se for outra condição, vem de outra forma.

Estrutura Básica

A cláusula `CASE` tem duas formas principais: a forma simples e a forma pesquisada.

1. **Forma Simples:**

```
CASE expressao
WHEN valor1THEN resultado1
WHEN valor2THEN resultado2
    ...
```

```
ELSE resultado_padrao
ENDCopiar código
```

Aqui, `expressao` é avaliada e comparada sequencialmente com cada `valorN`. Se uma correspondência é encontrada, o `resultadoN` correspondente é retornado.

2. **Forma Pesquisada:**

```
CASEWHEN condicao1THEN resultado1
WHEN condicao2THEN resultado2
   ...
ELSE resultado_padrao
ENDCopiar código
```

Nesta forma, cada `condicaoN` é uma expressão booleana. O `resultadoN` para a primeira condição verdadeira é retornado.

Exemplos Práticos

1. **Classificar dados em categorias:**
   - Suponha que você tenha uma tabela de `Pedidos` com uma coluna `TotalVenda`. Você quer classificar cada venda em 'Baixa', 'Média' ou 'Alta':

```
SELECT PedidoID,
CASEWHEN TotalVenda < 100THEN 'Baixa'
WHEN TotalVendaBETWEEN 100AND 500THEN 'Média'
ELSE 'Alta'
ENDAS CategoriaVenda
FROM Pedidos;
Copiar código
```

2. **Aplicar cálculos condicionais:**
   - Digamos que você queira dar um desconto de 10% para pedidos acima de $500 e um desconto de 5% para pedidos entre $100 e $500:

```
SELECT PedidoID, TotalVenda,
CASEWHEN TotalVenda > 500THEN TotalVenda * 0.9
```

```
WHEN TotalVendaBETWEEN 100AND 500THEN TotalVenda * 0.95
ELSE TotalVenda
ENDAS TotalComDesconto
FROM Pedidos;
Copiar código
```

Considerações Importantes

- **Performance:** O uso excessivo de instruções `CASE` pode afetar a performance da consulta, especialmente em grandes conjuntos de dados.
- **Legibilidade:** Embora a cláusula `CASE` seja poderosa, consultas muito complexas podem se tornar difíceis de ler e manter.
- **Compatibilidade:** A cláusula `CASE` é amplamente suportada pela maioria dos SGBDs, tornando-a uma ferramenta versátil para análise de dados.

RENAME

ALTER TABLE /*Tabela desejada*/ RENAME TO /*novo nome desejado*/;

```
ALTER TABLE HistoricoEmprego RENAME TO CargosColaboradores;
```

> Serve para renormearmos determinada tabela para outro nome que desejamos.

1. Clareza e Descritividade:

- **Nomes Significativos:** Escolha nomes que reflitam claramente o conteúdo e a função da tabela ou coluna. Por exemplo, uma tabela que armazena informações sobre clientes pode ser chamada de `Clientes` ou `InformacoesClientes`.
- **Evite Abreviações Obscuras:** Abreviações podem tornar os nomes mais curtos, mas devem ser evitadas a menos que sejam amplamente compreendidas e consistentes em todo o banco de dados.

2. Consistência:

- **Convenção de Nomenclatura:** Escolha uma convenção de nomenclatura e seja fiel a ela em todo o banco de dados. Por exemplo, se você usar nomes no singular para tabelas (Cliente), use isso consistentemente. O mesmo

vale para a capitalização; escolha entre PascalCase, camelCase ou snake_case e seja consistente.

- **Padrões de Nomenclatura de Coluna:** Mantenha um padrão para nomes de colunas semelhantes em diferentes tabelas. Por exemplo, se uma coluna que se refere a um identificador único é nomeada `IdCliente` em uma tabela, não a nomeie `ClienteID` em outra.

3. Evitar Palavras Reservadas:

- **Palavras do SQL:** Evite usar palavras reservadas do SQL como nomes de tabelas ou colunas, como `SELECT`, `DATE`, `TABLE`, etc. Isso pode causar conflitos e erros em consultas.

4. Precisão e Escopo:

- **Especificidade:** Nomes de colunas devem ser precisos. Por exemplo, em vez de chamar uma coluna de `Data`, nomeie-a como `DataNascimento` ou `DataContratacao`, dependendo do contexto.
- **Qualificação de Nomes:** Em um banco de dados com muitas tabelas relacionadas, pode ser útil incluir uma referência à tabela pai no nome de uma coluna de chave estrangeira. Por exemplo, `IdCliente` em uma tabela de pedidos.

5. Simplicidade e Tamanho:

- **Nomes Curtos, Mas Descritivos:** Enquanto a descritividade é importante, nomes excessivamente longos podem ser problemáticos para digitar e podem não ser totalmente suportados por todos os sistemas de banco de dados.

6. Utilizar Sublinhados para Espaços:

- **Sem Espaços:** Não use espaços em nomes de tabelas ou colunas. Use sublinhados (`_`) se necessário para separar palavras.

7. Documentação:

- **Mantenha Documentação:** Documente as convenções de nomenclatura e as decisões específicas de nomes para facilitar a compreensão e manutenção por outros usuários e desenvolvedores.

8. Idioma:

- **Considere o Idioma Padrão:** Use um idioma consistente (geralmente inglês) para nomes de tabelas e colunas, a menos que haja uma razão específica para não fazê-lo.